Bab 9

Kumpulan Contoh Kasus

Bab ini akan memberikan beberapa contoh SQL dalam berbagai aplikasi atau contoh-contoh kasus spesifik tertentu. Contoh-contoh aplikasi yang ada di sini bukanlah berupa contoh lengkap beserta program [PHP] melainkan hanya skema database dan query-query SQL yang umum dilakukan. Tujuan bab ini adalah untuk memperlihatkan bagaimana SQL bekerja dan bagaimana metode pendekatan dalam menyelesaikan sebuah query.

9.1 CMS

Tentang CMS

CMS atau *Content Management System* adalah istilah untuk sebuah aplikasi (pada umumnya berbasis web) yang mengizinkan pemakainya menyimpan dokumen (yang pada umumnya disimpan di database relasional) lalu menampilkan hasilnya (pada umumnya di website publik, tapi dapat juga di portal intranet).

CMS pada dasarnya berguna agar sebuah website dapat terus menambah/mengupdate kontennya (artikel, komentar artikel, polling, dan lain sebagainya) tanpa pengurusnya harus berurusan dengan kode HTML atau pemrograman atau transfer file FTP. CMS mempermudah orang yang awam komputer seperti wartawan/penulis/editor untuk memanage websitenya. Mereka cukup mengedit naskah via <TEXTAREA> atau editor rich text Javascript; mengeset beberapa atribut seperti kapan harus muncul di website, apa kategori artikel, dan lain sebagainya; lalu menekan Submit. Artikel akan tersimpan dan siap muncul di halaman depan (atau halaman-halaman dalam) website. Demikian juga dalam mengedit/menghapus naskah, mengedit kategori, menambah/mengedit poll, dan lain sebagainya. Tentu saja tidak semua jenis perubahan dapat dilakukan tanpa pemrograman atau perubahan layout, namun untuk masalah seputar konten saja semua ini dapat dilakukan lewat interface CMS.

Situs-situs berita dan portal seperti kompas.com, detik.com, dan lain sebagainya. praktis hampir selalu menggunakan CMS. Aplikasi web PHP seperti Mambo (http://www.mamboserver.com), PHPNuke (http://www.phpnuke.org), atau PostNuke (http://www.postnuke.org) termasuk dikategorikan ke dalam CMS. Walaupun PHPNuke dan PostNuke sifatnya lebih luas yakni ke portal namun salah satu komponen utamanya tetaplah manajemen artikel.

Skema Database

Tabel utama di CMS adalah tabel yang menyimpan daftar artikel. Artikel disimpan di dalam kolom TEXT atau MEDIUMTEXT atau LONGTEXT (bergantung apakah ingin membatasi teks hanya hingga 64KB saja, atau hingga 16MB atau bahkan hingga 4GB):

```
CREATE TABLE artikel (
  id INT PRIMARY KEY AUTO_INCREMENT,

  penulis VARCHAR(64) NOT NULL,
  id_user INT NOT NULL,
    INDEX(id_user),
```

```
    FOREIGN KEY (id_user) REFERENCES user(id),
  flagterbit BOOLEAN NOT NULL,
  flagkomentar BOOLEAN NOT NULL,
  tglterbit DATETIME,
    INDEX(tglterbit),
  tglbuat DATETIME NOT NULL,
    INDEX(tglbuat),
  tglubah DATETIME NOT NULL,
    INDEX(tglubah),
  catatan VARCHAR(255) NOT NULL,

  judul VARCHAR(150) NOT NULL,
  subjudul VARCHAR(150) NOT NULL,
  ringkasan TEXT,
  isi MEDIUMTEXT,
    FULLTEXT INDEX (judul, subjudul, ringkasan, isi),

  pathgambar VARCHAR(255) NOT NULL,
  paththumbnail VARCHAR(255) NOT NULL
);
```

Atribut terpenting yang perlu kita simpan untuk artikel adalah *judul*, *subjudul*, *ringkasan*, dan *isi*. *Subjudul* kadang disebut juga judul kedua atau judul kecil. Di surat kabar biasanya Anda bisa melihat judul dengan ukuran font yang lebih kecil di atas judul utama, itulah subjudul. *Ringkasan* kadang disebut juga summary atau preview atau *tease*, biasanya berupa satu atau dua paragraf awal artikel; ditampilkan di halaman depan atau halaman daftar artikel sebelum pengunjung mengklik untuk melihat isi artikel sepenuhnya. Kita membuat full-text index terhadap (*judul*, *subjudul*, *ringkasan*, *isi*) agar staf maupun pengunjung website nanti dapat melakukan search terhadap salah satu komponen konten artikel tersebut.

Berikutnya atribut-atribut metadata lain seperti *tglbuat* (kapan artikel pertama kali disubmit ke database), *tglubah* (tanggal terakhir modifikasi), *penulis* (nama dari satu atau lebih penulis), *flagkomentar* (apakah artikel boleh diberi komentar).

Artikel di database dapat berada dalam kondisi *terbit* atau *tidak terbit*, dan ini dicatat di *flagterbit*. Artikel yang terbit berarti dapat dilihat (dan di-search) oleh pengunjung sementara artikel yang tidak terbit artinya masih berupa draft dan belum dapat dilihat oleh pengunjung.

Untuk memudahkan pekerjaan wartawan, biasanya CMS memiliki fasilitas

autopublishing atau scheduled publishing. Artinya, kita bisa mengeset kapan (dicatat di *tglterbit*) sebuah artikel statusnya berubah menjadi dari tidak terbit menjadi terbit. Jadi seorang wartawan bisa memasukkan beberapa artikel di hari Jumat sore dan mengeset agar terbit otomatis hari Senin pagi. Si wartawan sendiri dapat datang Senin siang atau tidak datang sama sekali.

Atribut *catatan* adalah untuk catatan internal para staf di halaman administrasi. Pengunjung website tidak akan melihatnya.

```
CREATE TABLE kategori (
  id INT PRIMARY KEY AUTO_INCREMENT,
  path VARCHAR(255) NOT NULL,
    INDEX(path),
  nama VARCHAR(128) NOT NULL
);
```

Tabel ini mencatat daftar kategori dan kita buat sebagai tree dengan metode MP (lihat Bab 8).

```
CREATE TABLE artikel_kategori (
  id_artikel INT NOT NULL,
    INDEX(id_artikel),
    FOREIGN KEY (id_artikel) REFERENCES artikel(id),
  id_kategori INT NOT NULL,
    UNIQUE(id_kategori, id_artikel),
    FOREIGN KEY (id_kategori) REFERENCES kategori(id)
);
```

Tabel *artikel_kategori* mencatat hubungan M:M (many-to-many) antara artikel dan kategori. Sebuah artikel dapat dimasukkan ke dalam beberapa kategori sekaligus dan sebaliknya sebuah kategori pun dapat berisi banyak artikel.

```
CREATE TABLE komentar (
  id INT PRIMARY KEY AUTO_INCREMENT,
  tglbuat DATETIME NOT NULL,
    INDEX(tglbuat),
  tglubah DATETIME NOT NULL,
  id_artikel INT NOT NULL,
    INDEX(id_artikel),
    FOREIGN KEY (id_artikel) REFERENCES artikel(id),
  id_user INT NOT NULL,
```

```
    INDEX(id_user),
    FOREIGN KEY (id_user) REFERENCES user(id),

  judul VARCHAR(150) NOT NULL,
  isi TEXT,
    FULLTEXT INDEX (judul, isi)
);
```

Tabel *komentar* berisi komentar. Setiap komentar terasosiasi dengan sebuah artikel (melalui atribut *id_artikel*). Jika Anda menginginkan komentar dapat berbentuk thread, maka Anda bisa membuat tabel *komentar* ini menjadi tree dan menambahkan atribut *path*. Setiap balasan atas sebuah komentar ditaruh sebagai child node dari komentar yang dibalas. Setiap komentar ditulis oleh seorang user, dan daftar user dicatat di tabel *user*.

```
CREATE TABLE user (
  id INT PRIMARY KEY AUTO_INCREMENT,
  email VARCHAR(255) NOT NULL,
    UNIQUE(email),
  nama VARCHAR(150) NOT NULL,

  hashpassword CHAR(36) NOT NULL,

  tglbuat DATETIME NOT NULL,
  tglubah DATETIME NOT NULL,

  alamat1 VARCHAR(150) NOT NULL,
  alamat2 VARCHAR(150) NOT NULL,
  kota VARCHAR(64) NOT NULL,
  propinsi VARCHAR(32) NOT NULL,
  negara VARCHAR(32) NOT NULL,

  telp VARCHAR(32) NOT NULL,
  fax VARCHAR(32) NOT NULL,
  hp VARCHAR(32) NOT NULL,

  uin VARCHAR(16) NOT NULL, – nomor ICQ
  yahooid VARCHAR(32) NOT NULL, – Yahoo! Messenger

  flaglogin BOOLEAN NOT NULL DEFAULT 1,
  flagloginbackend BOOLEAN NOT NULL,
```

```
  flageditartikel BOOLEAN NOT NULL
  – ... flag-flag lain, jika ada
);
```

User dapat berupa pengunjung website yang telah mendaftar dan dapat pula berupa staf website. *tglbuat* adalah tanggal si user dibuat atau mendaftar. Kolom *email* dibuat unik agar seorang user dapat teridentifikasi berdasarkan alamat emailnya dan alamat email bisa dipakai sebagai username untuk login. Anda juga dapat menambahkan kolom unik username jika ingin user login menggunakan username buatan yang berbeda dengan email. *flaglogin* menyatakan apakah si member dapat login. Member yang sedang disuspend dapat diset nilai flagnya menjadi 0. Staf umumnya memiliki juga nilai *flagloginbackend* dan *flageditartikel* 1, artinya dapat masuk ke halaman administrasi dan dapat mengedit artikel. Anda dapat menambahkan atribut-atribut permission lain untuk mengatur hak tiap user, misalnya *flagcomment* untuk mengatur apakah dapat mempost komentar atau tidak, *flagpoll* untuk mengatur apakah dapat membuat/mengedit poll atau tidak, dan lain sebagainya.

Sebagai catatan, di tabel *artikel* kita membuat atribut teks *penulis* dan tidak *id_user* untuk mengambil nama user dari tabel *user* saja, karena kadang penulis memiliki nama alias atau kadang sebuah artikel tidak ingin ditulis nama penulisnya (diset kosong). Dengan atribut *penulis* yang berupa VARCHAR kita memiliki kebebasan untuk mengeset nama penulis ingin ditulis seperti apa.

```
CREATE TABLE poll (
  id INT PRIMARY KEY,
  pertanyaan VARCHAR(255) NOT NULL,

  tglbuat DATETIME NOT NULL,
    INDEX(tglbuat),

  flagtutup BOOLEAN NOT NULL,
  flaglihatpublik BOOLEAN NOT NULL,
  flaglihatdulu BOOLEAN NOT NULL DEFAULT 1
  – ... flag-flag lain jika ada
);

CREATE TABLE pilihanpoll (
  id_poll INT NOT NULL,
    INDEX(id_poll),
    FOREIGN KEY (id_poll) REFERENCES poll(id),
```

```
  pilihan VARCHAR(150) NOT NULL,
  jumlah INT NOT NULL,
  urutan INT NOT NULL,
  UNIQUE(urutan, id_poll)
);
```

Untuk polling, dibuat 2 tabel yaitu *poll* dan *pilihanpoll* dengan hubungan 1:M (one-to-many). Kolom *jumlah* menyatakan counter untuk tiap pilihan poll, sementara urutan untuk menyatakan urutan tampilan pilihan.

Halaman Depan/Halaman Daftar Artikel

Sekarang kita akan melihat seperti apa query-query yang sering digunakan di CMS. Untuk menampilkan daftar artikel di halaman depan, misalnya 10 artikel terbaru:

```
-- Contoh 9-1-1: Menampilkan daftar artikel
SELECT
  id, tglbuat, penulis, judul, subjudul, ringkasan
FROM artikel
WHERE flagterbit=1
ORDER BY tglbuat DESC
LIMIT 10;
```

Kita menambahkan klausa `WHERE flagterbit=1` untuk mencegah artikel-artikel draft ikut muncul. Untuk atribut kita hanya mengambil beberapa atribut saja. Atribut seperti *isi* tidak perlu diambil karena toh di halaman daftar artikel tidak akan ditampilkan dulu isinya. Kita perlu mengambil atribut *id* agar nanti dapat membuat link ke artikel ybs (mis: `isi_artikel.php?id=123`).

Untuk menampilkan semua artikel 1 minggu terakhir, Anda tinggal membuat klausa LIMIT dan mengganti klausa WHERE menjadi:

```
WHERE flagterbit=1 AND tglbuat >= DATE_SUB(NOW(), INTERVAL 7 DAY);
```

Untuk menampilkan hanya artikel yang memiliki kategori 'Berita Politik' saja, atau jika kita ingin mengambil nama kategori, kita perlu melakukan join dengan tabel *kategori* dan *artikel_kategori*:

```
-- Contoh 9-1-2: Menampilkan daftar artikel beserta kategorinya
SELECT
  a.id AS id,
  a.tglbuat AS tglbuat,
```

```
  penulis,
  k.nama AS namakategori,
  judul, subjudul, ringkasan
FROM artikel a
JOIN artikel_kategori ak ON ak.id_artikel = a.id
JOIN kategori k ON ak.id_kategori = k.id
WHERE flagterbit=1 AND k.nama = 'Berita Politik';
```

Jika Anda ingin mengambil semua kategori, cukup hilangkan pengujian `k.nama = 'Berita Politik'` di klausa WHERE.

Daftar Kategori

Untuk menampilkan daftar kategori, cukup lakukan SELECT sederhana ke tabel kategori. Anda dapat mengindentasi subkategori dengan query seperti ini:

```
-- Contoh 9-1-3: Menampilkan daftar kategori dengan indentasi
SELECT
  id, path,
  CONCAT(REPEAT('  ', CHAR_LENGTH(path)/4), nama) AS nama
FROM kategori
ORDER BY CONCAT(
  path,
  IF(id>999,'',IF(id>99,'0',IF(id>9,'00','000'))), id
);
```

Sehingga hasilnya seperti di bawah:

```
+----+----------+----------------------+
| id | path     | nama                 |
+----+----------+----------------------+
|  3 |          | Berita               |
|  4 | 0003     |   Berita Politik     |
|  5 | 0003     |   Berita Ekonomi     |
|  6 | 0003     |   Berita Hiburan     |
|  7 | 0003     |   Berita Internasional |
|  8 | 0003     |   Berita Daerah      |
|  9 |          | Opini                |
| 11 | 0009     |   Surat Pembaca      |
| 14 | 0009     |   Kolom Pakar        |
| 12 | 00090014 |     Kolom Sutan Sinar  |
| 13 | 00090014 |     Kolom Rini Suwanto |
```

```
| 10 |          | Profil                 |
+----+----------+------------------------+
12 rows in set (0.03 sec)
```

Kita menggunakan fungsi REPEAT(*str*, *n*) untuk membentuk string yang terdiri dari *n* kali penggabungan *str*. Dalam contoh kita kali ini, kita ingin membuat deretan spasi yang semakin panjang seiring *path* yang semakin dalam. Kita melakukan pengurutan berdasarkan *path* + *id* agar sibling nodes (node yang parent/path-nya sama) bisa ditampilkan berdekatan. Karena di MySQL dan rata-rata database lain tidak terdapat fungsi `SPRINTF("%04d", id)` maka kita menggunakan serangkaian IF() untuk membuat tampilan *id* menjadi konstan 4 digit dengan prefiks 0 opsional.

Tentu saja indentasi bisa juga dilakukan di sisi bahasa pemrograman, misalnya dengan fungsi `str_repeat()` di PHP.

Halaman Isi Artikel

Untuk menampilkan isi artikel, cukup melakukan SELECT sederhana dari tabel `artikel`. Yang perlu diingat di sini adalah tetap menambahkan klausa `WHERE flagterbit=1` agar jika pembaca memasukkan URL berisi id artikel yang belum terbit, hasilnya adalah tidak ditemukan.

Halaman Pencarian

Untuk mencari artikel berdasarkan kata kunci tertentu, kita cukup melakukan SELECT sama seperti sewaktu menampilkan halaman daftar artikel, hanya saja kita menambahkan kondisi:

```
WHERE ... MATCH(judul, subjudul, ringkasan, isi)
          AGAINST('kata kunci')
```

di mana `'kata kunci'` adalah kata kunci yang dimasukkan oleh pengunjung.

Poll

Untuk menampilkan sebuah polling secara acak, pertama-tama kita memilih id dan pertanyaan poll dari tabel *poll*:

```
-- Contoh 9-1-4: Memilih poll secara acak
-- Catatan: hanya pilih poll yang masih terbuka/masih boleh divote
SELECT id, pertanyaan
FROM poll
WHERE flagtutup=0
ORDER BY RAND()
```

```
LIMIT 1;
```

Kemudian kita ambil daftar pertanyaan untuk poll tersebut:

```
-- Contoh 9-1-5: Mengambil daftar pilihan poll
SELECT urutan, pilihan
FROM pilihanpoll
WHERE id_poll=... — masukkan id poll yang telah diambil
sebelumnya
ORDER BY urutan;
```

Jika ada yang voting, kita tinggal meningkatkan nilai counter:

```
UPDATE pilihanpoll SET jumlah=jumlah+1
WHERE urutan=... AND id_poll=...; — masukkan id dan urutan
pilihan ybs
```

Pada umumnya, skrip mengecek dulu berdasarkan IP dan/atau cookie agar pengunjung yang nakal tidak melakukan voting berulang-ulang untuk id poll yang sama. Jika kita melakukan pengecekan berdasarkan alamat IP, maka kita perlu membuat tabel yang merekam tiap IP dan id poll.

Halaman Login

Kolom *hashpassword* di tabel user bertipe CHAR(36) dan menggunakan skema 4 digit heksadesimal salt + 32 digit heksadesimal hash MD5 (salt + password). Jadi kita tidak menyimpan password mentah user, demi privasi si user. Berikut ini potongan kode PHP untuk mengeset password user:

```
<?php
$password = "rahasia";
$salt = sprintf("%04x", rand(0, 65535)); # buat salt
$hashpassword = $salt . md5($salt . $password); # bentuk hash
?>
```

Nilai hash inilah yang kita masukkan ke database, bukan passwordnya itu sendiri. Lalu bagaimana mengecek apakah user "`steven@masterwebnet.com`" memasukkan password yang benar yaitu "`rahasia`"?

```
-- Contoh 9-1-6: Mencocokkan password user
SELECT
  (hashpassword =
```

```
    -- buat hash dengan salt dari database dan password dari input
    CONCAT(
      SUBSTRING(hashpassword,1,4),
      MD5(CONCAT(SUBSTRING(hashpassword,1,4),'rahasia'))
    )
  ) AS cocok
FROM user
WHERE email = 'steven@masterwebnet.com';
```

Jika kolom hasil *cocok* bernilai 1, maka password yang dimasukkan benar karena ketika dicoba dibuatkan hashnya, hasilnya cocok dengan nilai *hashpassword* di database.

Halaman Back End

Halaman backend (halaman administrasi bagi staf untuk memanage website) berisi halaman login yang sama seperti halaman login publik (dengan tambahan pengecekan *fieldloginbackend* harus bernilai 1). Selain itu ada halaman untuk mendaftar artikel, mengisi artikel, mengedit artikel, dan lain sebagainya. Semua querynya pada intinya merupakan kombinasi SELECT, INSERT, UPDATE, dan DELETE saja.

Scheduled Publishing

Untuk melakukan scheduled publishing, kita dapat membuat skrip yang dijalankan misalnya setiap 5 menit dari crontab Unix atau Task Scheduler Windows. Skrip cukup konek ke database dan melakukan query berikut:

```
-- Contoh 9-1-7: Scheduled publishing
UPDATE artikel SET flagterbit=1
WHERE
  flagterbit=0 AND
  tglterbit IS NOT NULL AND
  tglterbit != '0000-00-00 00:00:00' AND
  tglterbit <= NOW();
```

Query di atas mencari semua artikel yang belum terbit (*flagterbit=0*) dan *tglterbit*-nya diset dan telah lewat dari waktu saat ini, lalu mengubah *flagterbit* menjadi 1.

Latihan

Sebagai latihan untuk Anda, cobalah melakukan query-query berikut ini:

1. User mana yang paling banyak menulis artikel?
2. Artikel mana yang paling popular (paling banyak komentarnya)?

3. Bagaimana mencari artikel berdasarkan kata kunci yang dikandungnya atau yang dikandung dalam salah satu komentar untuk artikel tersebut?

9.2 Blog

Sejak beberapa tahun yang lalu *blogger.com* memperkenalkan istilah "blog" maka kini telah berkembang ratusan ribu (jutaan?) blogger yang hampir setiap hari menulis diari atau jurnalnya di Web.

Model database dan query-query SQL untuk blog sebetulnya mirip dengan CMS, karena sebuah entri blog pada dasarnya adalah artikel. Blog pun sama seperti artikel CMS dapat diberi komentar. Dan ada kategori-kategori agar si blogger dapat mengentri jurnal pribadi atau jurnal teknis secara terpisah.

Fitur yang spesifik/unik yang biasa dijumpai pada blog adalah adanya uji CAPTCHA pada penulisan komentar untuk mencegah robot spam serta fasilitas linking/tracking agar jika seorang blogger mengomentari blog milik orang lain, si blog yang dikomentari dapat diberi tahu (di-"ping") oleh skrip si blog pengomentar dan akan muncul otomatis di blog si yang dikomentari. Ini tidak membutuhkan query SQL khusus dan tidak perlulah kiranya dibahas di dalam buku ini.

9.3 Web Board

Web board atau web forums tentunya sudah tidak asing bagi para pengguna Internet. Aplikasi PHP seperti vBulletin *(http://www.vbulletin.org)* dan phpBB *(http://www.phpbb.com)* merupakan dua contoh dari sekian banyak web board yang tersedia.

Sebuah web board umumnya terdiri dari beberapa kategori 1 level. Setiap kategori dapat diisi dengan beberapa forum. Setiap forum dapat diisi dengan topik-topik. Dan setiap topik dapat diisi dengan posting.

Berikut ini skema dasar database web board:

```
CREATE TABLE kategoriforum (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(150) NOT NULL,
  deskripsi TEXT,
  urutan INT NOT NULL,
    UNIQUE(urutan)
);
```

```
CREATE TABLE forum (
  id INT PRIMARY KEY AUTO_INCREMENT,
  nama VARCHAR(150) NOT NULL,
  deskripsi TEXT,
  tglbuat DATETIME NOT NULL,

  urutan INT NOT NULL,
  id_kategori INT NOT NULL,
    INDEX(id_kategori),
    FOREIGN KEY (id_kategori) REFERENCES kategoriforum(id),
  UNIQUE(id_kategori, urutan)
);

CREATE TABLE topik (
  id INT PRIMARY KEY AUTO_INCREMENT,
  id_forum INT NOT NULL,
    INDEX(id_forum),
    FOREIGN KEY (id_forum) REFERENCES forum(id),
  judul VARCHAR(150) NOT NULL,
  isi TEXT,
    FULLTEXT INDEX (judul, isi),
  id_user INT NOT NULL,
    INDEX(id_user),
    FOREIGN KEY (id_user) REFERENCES user(id),
  tglbuat DATETIME NOT NULL,
    INDEX(tglbuat),
  tglubah DATETIME NOT NULL,
    INDEX(tglubah)
);

CREATE TABLE post (
  id INT PRIMARY KEY AUTO_INCREMENT,
  id_topik INT NOT NULL,
    INDEX(id_topik),
    FOREIGN KEY (id_topik) REFERENCES topik(id),
  judul VARCHAR(150) NOT NULL,
  isi TEXT,
    FULLTEXT INDEX (judul, isi),
  id_user INT NOT NULL,
    INDEX(id_user),
```

```
    FOREIGN KEY (id_user) REFERENCES user(id),
  tglbuat DATETIME NOT NULL,
    INDEX(tglbuat),
  tglubah DATETIME NOT NULL,
    INDEX(tglubah)
);
```

Tabel *user* tidak saya sebutkan di sini karena kurang lebih mirip dengan tabel user pada aplikasi lain seperti CMS. Tentunya dalam kenyataannya, database web board tidak hanya terdiri dari hanya 5 tabel ini saja. Setiap forum umumnya dapat diberi permission tertentu, jadi forum tertentu hanya bisa dilihat oleh user tertentu saja sementara forum lain bersifat publik dan dapat dilihat oleh siapa saja termasuk pengunjung yang belum login. Jumlah posting setiap user juga umumnya direkam di database (agar tidak perlu diquery terus via `SELECT COUNT(*) ...`) dan kadang jumlah total posting si user menentukan "status" atau "rankingnya". Selain itu, biasanya aplikasi forum juga memiliki fasilitas seperti private messaging.

Halaman Depan

Halaman depan web board berisi daftar forum yang ditampilkan sesuai kategori dan urutan forum itu sendiri dalam sebuah kategori, beserta dengan informasi jumlah topik dan total post yang ada dalam sebuah forum. Query SQL-nya kurang lebih seperti ini:

```
-- Contoh 9-3-1: Menampilkan daftar forum
SELECT
  id, nama, deskripsi,
  (SELECT COUNT(*) FROM topik WHERE id_forum=forum.id) AS
jum_topik,
  (SELECT COUNT(*) FROM post WHERE id_topik IN
    (SELECT id FROM topik WHERE id_forum=forum.id)) AS jum_post
FROM forum
ORDER BY id_kategori, urutan;
```

Sementara untuk mengambil daftar kategorinya sendiri:

```
-- Contoh 9-3-2: Menampilkan daftar kategori
SELECT nama, deskripsi
FROM kategori ORDER BY urutan;
```

Selanjutnya tinggal skrip PHP Anda menampilkan halaman HTML kategori berdasarkan urutan kategori, dan daftar forum yang ada di setiap kategori, sesuai

urutan forum dalam kategori tersebut.

Halaman Forum

Jika pengunjung mengklik salah satu forum, maka ia akan disuguhi halaman forum yang berisi daftar topik yang ada dalam forum tersebut:

```
-- Contoh 9-3-3: Menampilkan daftar topik dalam sebuah forum
SELECT
  id, judul,
  (SELECT COUNT(*) FROM post WHERE id_topik=topik.id) AS
jum_post,
                        – jum_post kadang disebut pula jum_reply
  COALESCE(
    (SELECT UNIX_TIMESTAMP(MAX(tglbuat)) FROM post
      WHERE id_topik=topik.id),
    UNIX_TIMESTAMP(tglbuat)
  ) AS tgl_post_terbaru
FROM topik
WHERE id_forum=1 – masukkan id forum sebenarnya di sini
ORDER BY tgl_post_terbaru DESC;
```

Di sini saya menggunakan UNIX_TIMESTAMP() untuk tanggal agar setelah diambil di PHP, Anda bisa bebas menggunakan fungsi PHP `date()` untuk memformat tanggal sesuai keinginan.

Perhatikan ekspresi COALESCE(...). Pertama kita mencari tanggal post terbaru untuk sebuah topik. Namun jika sebuah topik belum ada yang membalas, maka tidak akan ada barisnya di tabel *post* dan hasil dari subquery adalah NULL. Dalam kasus tersebut, kita memilih *tglbuat* si topik itu sendiri. Sehingga kita dapat mengurutkan tampilan daftar topik berdasarkan tanggal post terakhir atau tanggal si topik itu sendiri dibuat (untuk topik yang belum ada balasan).

Halaman Topik

Jika pengunjung mengklik salah satu topik, maka ia akan sampai pada halaman isi topik. Isi dari halaman ini adalah isi topik dan isi posting-posting yang membalasnya, dalam urutan tanggal mulai dari yang lebih awal.

```
-- Contoh 9-3-4: Menampilkan daftar posting untuk sebuah topik
SELECT id, tglbuat, tglubah, judul, isi
FROM topik
WHERE id =1 – ganti dengan id topik sebetulnya
```

```
UNION ALL

SELECT id, tglbuat, tglubah, judul, isi
FROM post WHERE id_topik=1 – ganti dengan id topik sebetulnya


ORDER BY tglbuat ASC;
```

Paging

Bagaimana seandainya sebuah topik amat ramai dan memiliki banyak posting? Umumnya kita melakukan paging agar pengunjung tidak terlalu harus membuka halaman HTML yang amat besar ukurannya.

Untuk melakukan paging, pertama kita harus mengetahui ada berapa posting untuk sebuah topik? Catatan: kita menambahkan satu untuk si topik itu sendiri (contoh: jika sebuah topik memiliki 2 posting balasan maka total jumlah posting adalah 1 + 2 = 3).

```
SELECT COUNT(*) + 1 FROM post
WHERE id_topik=1; – ganti dengan id topik yang sebetulnya
```

Simpanlah nilai ini ke dalam variabel, misalnya `$num_rows`. Saat ingin menampilkan sebuah halaman topik, kita cek dulu variabel request GET `$page`. Jika nilainya tidak terdefinisi, kita asumsikan nomor halaman adalah 1. Kita lalu menghitung nilai `$offset` dengan `$pagesize*($page-1)`. `$pagesize` bisa kita preset misalnya dengan nilai 15 (15 posting per halaman topik). Sekarang kita tinggal menambahkan klausa LIMIT pada query. Misalnya untuk menampilkan halaman ketiga:

```
-- Contoh 9-3-5: Menampilkan daftar posting untuk sebuah topik
--   (dengan paging)
SELECT id, tglbuat, tglubah, judul, isi
FROM topik
WHERE id =1 – ganti dengan id topik sebetulnya

UNION ALL

SELECT id, tglbuat, tglubah, judul, isi
FROM post WHERE id_topik=1 – ganti dengan id topik sebetulnya

```

```
ORDER BY tglbuat ASC
LIMIT 30, 15;
```

Latihan

Cobalah menyusun query-query untuk melakukan hal berikut:

1. Topik mana yang paling popular (paling banyak postingnya)?
2. Berapa buah topik baru dan posting baru yang dibuat selama seminggu terakhir?
3. User mana yang paling aktif memposting di web board? User mana yang paling aktif memposting di forum A?
4. Carilah topik yang mengandung kata kunci 'orang utan', menggunakan full-text index.
5. Posting mana saja yang sudah pernah diedit? Tampilkan judul topik dan nama forum tempat topik tersebut berada.

9.4 Medali Olimpiade

Contoh kasus ini untuk mendemonstrasikan custom ordering (pengurutan berdasarkan ekspresi) dan penggunaan subquery.

Seperti kita ketahui, Olimpiade musim panas 2004 di Athena dilaksanakan bulan Agustus lalu dan hasilnya adalah seperti di bawah (diringkas; lengkapnya dapat Anda ambil dari file `code/olimpiade.sql` di CD buku). Ada 75 negara yang berhasil mendapatkan medali. Indonesia berada di peringkat ke-48.

```
+------+----------------+------+-------+----------+-------+---------+
| kode | nama           | emas | perak | perunggu | total | ranking |
+------+----------------+------+-------+----------+-------+---------+
| USA  | United States  |   35 |    39 |       29 |   103 |       1 |
| CHN  | China          |   32 |    17 |       14 |    63 |       2 |
| RUS  | Russia         |   27 |    27 |       38 |    92 |       3 |
| AUS  | Australia      |   17 |    16 |       16 |    49 |       4 |
| JPN  | Japan          |   16 |     9 |       12 |    37 |       5 |
| GER  | Germany        |   14 |    16 |       18 |    48 |       6 |
| FRA  | France         |   11 |     9 |       13 |    33 |       7 |
| ITA  | Italy          |   10 |    11 |       11 |    32 |       8 |
| KOR  | Korea          |    9 |    12 |        9 |    30 |       9 |
| GBR  | Great Britain  |    9 |     9 |       12 |    30 |      10 |
---
```

```
| SUI  | Switzerland     |    1 |     1 |         3 |     5 |       46 |
| EGY  | Egypt           |    1 |     1 |         3 |     5 |       46 |
| INA  | Indonesia       |    1 |     1 |         2 |     4 |       48 |
| ZIM  | Zimbabwe        |    1 |     1 |         1 |     3 |       49 |
---
| COL  | Colombia        |    0 |     0 |         1 |     1 |       71 |
| ERI  | Eritrea         |    0 |     0 |         1 |     1 |       71 |
| MGL  | Mongolia        |    0 |     0 |         1 |     1 |       71 |
| SYR  | Syrian Arab Rep |    0 |     0 |         1 |     1 |       71 |
| TRI  | Trinidad/Tobago |    0 |     0 |         1 |     1 |       71 |
+------+-----------------+------+-------+-----------+-------+----------+
75 rows in set (0.03 sec)
```

Untuk menentukan ranking, dasar perhitungannya adalah jumlah emas. Jika jumlah emas sama, maka yang dilihat adalah jumlah perak. Jika jumlah perak sama, dilihat jumlah perunggu.

Yang membuat rumit, bagaimana jika jumlah perunggu juga sama? Contoh kasusnya Swiss (SUI) dan Mesir (EGY). Apakah yang satu negara mau mengalah dan menjadi ranking 47 sementara yang satu lagi 46? Tentu tidak mau. Jadi untuk kasus perolehan medali yang sama, rankingnya juga sama. Dalam kasus ini sama-sama 46. Negara yang di bawah Swiss dan Mesir, yaitu negara kita, menempati urutan 48 (bukan 47), karena di atas Indonesia ada 47 negara.

Sekarang pertanyaannya, bagaimana query untuk menghasilkan hasil seperti di atas tersebut? Struktur tabelnya sendiri seperti ini:

```
CREATE TABLE negara (
  kode CHAR(3) PRIMARY KEY,
nama VARCHAR(32) NOT NULL
);

CREATE TABLE medali (
  kodenegara CHAR(3) PRIMARY KEY,
    INDEX(kodenegara),
    FOREIGN KEY (kodenegara) REFERENCES negara(kode),
  emas SMALLINT NOT NULL,
  perak SMALLINT NOT NULL,
  perunggu SMALLINT NOT NULL
);
```

Tabel *negara* berisi daftar negara sementara tabel *medali* mencatat perolehan medali tiap negara.

Jawabannya: kita harus menghitung ranking sebagai "(jumlah negara yang perolehan medalinya berada di atas negara tersebut) + 1". Contohnya, di atas Indonesia ada 47 negara yang perolehan medalinya lebih baik dari kita maka ranking Indonesia adalah 47+1 = 48. Swiss dan Mesir sama-sama memiliki 45 negara yang lebih baik peroleh medalinya, maka keduanya sama-sama beranking 45+1 = 46. Amerika tidak memiliki negara yang perolehannya lebih baik, maka rankingnya = 0 + 1 = 1.

Berikut ini querynya.

```
-- Contoh 9-4-1: Menentukan ranking negara di Olimpiade
SELECT
  kodenegara AS kode, nama,
  emas, perak, perunggu,
  emas+perak+perunggu AS total,

  (SELECT COUNT(*) FROM medali m1 WHERE
    m1.emas > m2.emas OR
    (m1.emas = m2.emas AND (m1.perak > m2.perak OR
      (m1.perak = m2.perak AND m1.perunggu > m2.perunggu))
    )
  )+1 AS ranking

FROM medali m2
LEFT JOIN negara ON kodenegara=negara.kode
GROUP BY emas, perak, perunggu
ORDER BY ranking;
```

Subquery yang dicetak tebal adalah yang kita gunakan untuk mencari jumlah negara yang perolehan medalinya lebih baik. Ini dilakukan untuk setiap negara, jadi seperti inner loop.

Latihan

Bagaimana query untuk menghasilkan tampilan seperti di bawah?

```
+------+----------------+------+-------+----------+-------+---------+
| kode | nama           | emas | perak | perunggu | total | ranking |
+------+----------------+------+-------+----------+-------+---------+
| USA  | United States  |   35 |    39 |       29 |   103 |       1 |
| CHN  | China          |   32 |    17 |       14 |    63 |       2 |
| RUS  | Russia         |   27 |    27 |       38 |    92 |       3 |
...
| SUI  | Switzerland    |    1 |     1 |        3 |     5 |     =46 |
| EGY  | Egypt          |    1 |     1 |        3 |     5 |     =46 |
| INA  | Indonesia      |    1 |     1 |        2 |     4 |      48 |
| ZIM  | Zimbabwe       |    1 |     1 |        1 |     3 |      49 |
...
| COL  | Colombia       |    0 |     0 |        1 |     1 |     =71 |
| ERI  | Eritrea        |    0 |     0 |        1 |     1 |     =71 |
| MGL  | Mongolia       |    0 |     0 |        1 |     1 |     =71 |
| SYR  | Syrian Arab Rep |    0 |     0 |        1 |     1 |     =71 |
| TRI  | Trinidad/Tobago |    0 |     0 |        1 |     1 |     =71 |
+------+----------------+------+-------+----------+-------+---------+
75 rows in set (0.03 sec)
```

Dengan kata lain, kita menambahkan simbol "=" jika sebuah ranking yang sama dimiliki oleh lebih dari satu negara.

9.5 Friendster

Tahun 2004 ini ada wabah tren yang sedang menggejala yaitu Friendster, http://www.friendster.com/, dan deretan klon-klonnya. Inti dari situs networking seperti ini adalah agar kita dapat melihat dan berkenalan dengan *teman dari teman* kita (dan temannya, dan seterusnya).

Skema dasar database ala Friendster cukup sederhana, yaitu tabel *user* (yang mirip dengan tabel *user* pada CMS atau web board) sebagai komponen utamanya. Lalu untuk mencatat apakah seseorang (si A) sudah berteman dengan orang lain (si B), kita membuat tabel *user_friend*:

```
CREATE TABLE user_friend (
  tglset DATETIME NOT NULL,
  id_user INT NOT NULL,
    INDEX(id_user),
    FOREIGN KEY (id_user) REFERENCES user(id),
```

```
  id_friend INT NOT NULL,
    UNIQUE(id_friend, id_user),
    FOREIGN KEY (id_friend) REFERENCES user(id)
    – CHECK (id_friend <> id_user)
);

-- contoh insert
INSERT INTO user_friend(NOW(), A, B);
```

Kita bisa juga menambah atribut lain seperti jenis pertemanan pada tabel *user_friend* ini (misalnya hanya kenal, pernah bertemu, atau teman dekat a la di Orkut, (http://www.orkut.com).

Lalu untuk mencari teman langsung dari seseorang (A), yang sering disebut *1st degree friend*:

```
-- Contoh 9-5-2: Mencari teman dari seseorang
SELECT
  id_friend, nama, email
FROM user_friend
LEFT JOIN user ON id_friend = user.id
WHERE id_user = A;
```

Untuk mencari teman dari teman seseorang (A), yang sering disebut *2nd degree friend*:

```
-- Contoh 9-5-2: Mencari teman dari teman dari seseorang
SELECT
  id_friend, nama, email
FROM user_friend
LEFT JOIN user ON id_friend = user.id
WHERE id_user IN
  (SELECT id_friend FROM user_friend WHERE id_user = A);
```

Untuk mencari 3rd degree friend kita tinggal menambah level subquery:

```
-- Contoh 9-5-3: Mencari teman dari teman dari teman dari seseorang
SELECT
  id_friend, nama, email
FROM user_friend
LEFT JOIN user ON id_friend = user.id
```

```
WHERE id_user IN
  (SELECT id_friend FROM user_friend WHERE id_user IN
    (SELECT id_friend FROM user_friend WHERE id_user = A)
  );
```

Untuk mendaftarkan semua teman dari tingkat 1 (1st degree) hingga tingkat 3 (2nd degree), kita bisa melakukan UNION dari ketiga SELECT di atas.